APPLICATION

FOR

UNITED STATES LETTERS PATENT

APPLICANT NAME: Giora Biran et al.

TITLE: PARALLEL TCP SENDER IMPLEMENTATION

DOCKET NO.: FIS920030298US1

INTERNATIONAL BUSINESS MACHINES CORPORATION

CERTIFICATE OF MAILING UNDER 37 CFR 1.10

I hereby certify that, on the date shown below, this correspondence is being deposited with the United States Postal Service in an envelope addressed to: Mail Stop Patent Application, Commissioner for Patents, P.O. Box 1450, Alexandria VA 22313-1450 as "Express Mail Post Office to Addressee" Mailing Label No. EV 108 089 602 US

on December 1, 2003

Jennifer Desbiens
Name of person mailing paper

Jennifer Desbiens 12/1/2003
Signature Date

PARALLEL TCP SENDER IMPLEMENTATION

BACKGROUND OF THE INVENTION

1. Technical Field

[0001] The present invention relates generally to a TCP (transmission control protocol) transmission system, and more specifically relates to a parallel implementation of a TCP sender comprising a transmit request handler and a transmitter.

2. Related Art

[0002] TCP (Transmission Control Protocol) is a connection-oriented transport protocol that sends data as an unstructured stream of bytes. By using sequence numbers and acknowledgment messages, TCP can provide a sending node with delivery information about packets transmitted to a destination node. Where data has been lost in transit from source to destination, TCP can retransmit the data until either a timeout condition is reached or until successful delivery has been achieved. TCP can also recognize duplicate messages and will discard them appropriately. If the sending computer is transmitting too fast for the receiving computer, TCP can employ flow control mechanisms to slow data transfer. TCP can also communicate delivery information to the upper-layer protocols and applications it supports.

[0003] TCP protocol was originally designed to be implemented as a single state machine (i.e., processing of events is serialized). However, as network transmission rates increase, straightforward serialization implementations fail to provide the required

performance, both for conventional processor-based implementations, and for hardware implementations. A typical sequential TCP transmission implementation builds appropriate packets during handling of different events, including: when an application posts a request for data transmission; when an acknowledgement arrives from a remote TCP; when data arrives from a remote TCP, triggering transmission of a data acknowledgement; and when one of the transmission, persist or delayed ACK timers expire.

[0004] In current implementations, the requester (i.e., producer of the event) must wait until event processing, including packet generation, is complete before the requester can continue its work. Unfortunately, this can cause a significant slowdown for the communication network. Accordingly, a need exists for a more efficient TCP transmission system that allows the requester to continue working after an event is produced.

SUMMARY OF THE INVENTION

[0005] The present invention addresses the above-mentioned problems as well as others, by providing parallel implementation of a TCP sender (i.e., transmission system) comprising a transmit request handler and a transmitter. In a first aspect, the invention provides a transmission control protocol (TCP) transmission system, comprising: a transmit request handler that receives request events, and either schedules a connection in a ready queue or places the connection in a pending queue; and a transmitter that operates in parallel with the transmit request handler, wherein the transmitter dequeues connections from the ready queue and prepares packets for transmission.

[0006] In a second aspect, the invention provides a method for transmitting packets in a transfer control protocol (TCP) environment, comprising: submitting a request event to a transmit request handler; processing the request event in the transmit request handler to either schedule a connection in a ready queue or place the connection in a pending queue; providing a transmitter that operates in parallel with the transmit request handler; and utilizing the transmitter to dequeue connections from the ready queue and prepare packets for transmission.

[0007] In a third aspect, the invention provides a system for transmitting packets in a transfer control protocol (TCP) environment, comprising: a connection context for storing event information; a transmit request handler that receives request events, records the event information into the connection context and either schedules a connection in a ready queue or places the connection in a pending queue; a transmitter that operates in parallel with the transmit request handler, wherein the transmitter dequeues connections from the ready queue and prepares packets for transmission based on event information stored in the connection context; and a queue manager for moving connections from the pending queue to the ready queue.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] These and other features of this invention will be more readily understood from the following detailed description of the various aspects of the invention taken in conjunction with the accompanying drawings in which:

[0009] Figure 1 depicts a TCP sender in accordance with the present invention.

[00010] Figure 2 depicts an exemplary flow of a TCP sender in accordance with the present invention.

DETAILED DESCRIPTION OF THE INVENTION

[00011] Referring now to the Figure 1, a parallel implementation of a TCP sender 10 is shown that includes a transmit request handler 12 and a transmitter 14. It is assumed for the purposes of this description that the reader has an understanding of transmission control protocol commensurate with one skilled in art of communication systems. Accordingly, a detailed description of TCP is not provided herein.

[00012] Transmit request handler 12 executes event requests 23 from a requester 22 for transmission of data and/or control packets. To allow for efficient parallel processing, transmit request handler 12 performs only minimal handling of requests 23, thereby significantly reducing the wait time for the requester 22. Instead of building packets to be transmitted (possibly at a much later time), transmit request handler 12 just "updates" the state of a corresponding connection. If transmit request handler 12 detects that the conditions for transmitting a packet are not yet satisfied, it can move the connection to a pending arbitration state or pending queue 16. Alternatively, if transmit request handler 12 detects that the conditions for transmitting a packet are satisfied, it schedules the connection to the transmit server by placing the connection in a ready arbitration state or ready queue 18.

[00013] Transmitter 14 fetches "ready" connections from the ready queue 18 in parallel to the operation of the transmit request handler 12. Specifically, transmitter 14 retrieves a ready connection, examines its context from connection context 20, builds the appropriate packet 24, and pushes it into the transmission FIFO 30. The only time the

transmit request handler 12 and transmitter do not operate in parallel is when they are both handling the same connection.

[00014] Context manager 26 manages the connection context 20 for each connection. Connection context 20 is essentially a data structure that stores data pertaining the particular connection. For instance, it includes bits, flags and/or data indicating: that a request has been made to send an immediate or delayed ACK; that a delayed ACK is to expire; that a transmission shutdown command occurred; how much ULP data is available; timer information; window information; retransmission information; etc.

[00015] Queue manager 28 handles requests from the transmit request handler 12 and transmitter 14 involving enqueue/dequeue operations. These processes are described below with reference to Figure 2.

[00016] Referring now to Figure 2, an exemplary data flow between the transmit request handler 12 and transmitter 14 is shown. As can be seen, ready queue 18 comprises a queue of connections scheduled for transmission by transmit request handler 12. The connections are maintained in the ready queue 18 as a single linked list (but may be implemented as multiple linked lists, or in any other known manner), using pointers kept in the connection context, and two registers 32 and 34 pointing to the head and tail of the list. Transmit request handler 12 enqueues connections that are ready to be served using the ready list tail 34, and transmitter 14 dequeues connections pointed by the ready list head 32.

[00017] As can be seen, transmit request handler 12 is equipped to handle all possible types of event requests, including: software start/shutdown commands 36; a

notification of new transmission data posted 38; a request to send an immediate ACK, delayed ACK, or window update 40; or handling an incoming ACK 42. An additional type of event is referred to as timer expiration 44, which is detected by queue manager 28. When this event occurs, the connection is moved to the ready queue 18, and eventually transmitter 14 performs the actual timeout processing. Described below are various examples of how transmit request handler 12 may handle requests 23.

[00018] The software start command (from commands 36) is used upon connection creation to trigger transmission of segments related to connection establishment (e.g., ACK), even though no data is available yet. When the transmit request handler 12 receives this command, it moves the connection to the ready queue 18.

[00019] When a notification of new transmission data 38 is posted, transmit request handler 12 updates a posted message count in the connection context 20 and performs a segmentation check to decide whether transmit request handler 12 should schedule data transmission. If transmission is allowed, or the connection is in urgent mode, transmit request handler 12 moves the connection to the ready queue 18. Otherwise, the connection is placed in the pending queue 16.

[00020] Requests to send an immediate ACK, delayed ACK, or window update 40 are handled as follows. Transmit request handler 12 moves the connection to the ready queue 18 if an immediate ACK was requested, or it too many bytes of data are unacknowledged. Transmit request handler 12 moves the connection to the pending queue 16 if the ACK should be delayed. When a request to send a window update is received, the appropriate connection context 20 is set and the connection is moved to the ready queue 18.

[00021] For an incoming ACK 40, transmit request handler 12 can perform various functions based on the information in the ACK, including: issuing a notification on completion of data transfer; requesting transmission of a new segment or of a segment according to a fast retransmit algorithm; start/stop the retransmit or persist timers; or issue a notification on arrival of an ACK.

[00022] Transmitter 14 provides numerous functions, including: handling timeouts of a retransmit timer; deciding the type of segment to transfer; building transmit commands and requesting segment data; building TCP, IP and MAC headers; recording information on the last transmitted segment (e.g., ACK number, Window, transmission time, etc.); starting the retransmit timer if data was transmitted; and deciding on a next arbitration state.

[00023] It is understood that the systems, functions, mechanisms, methods, engines and modules described herein can be implemented in hardware, software, or a combination of hardware and software. They may be implemented by any type of computer system or other apparatus adapted for carrying out the methods described herein. A typical combination of hardware and software could be a general-purpose computer system with a computer program that, when loaded and executed, controls the computer system such that it carries out the methods described herein. Alternatively, a specific use computer, containing specialized hardware for carrying out one or more of the functional tasks of the invention could be utilized. The present invention can also be embedded in a computer program product, which comprises all the features enabling the implementation of the methods and functions described herein, and which - when loaded in a computer system - is able to carry out these methods and functions. Computer

program, software program, program, program product, or software, in the present context mean any expression, in any language, code or notation, of a set of instructions intended to cause a system having an information processing capability to perform a particular function either directly or after either or both of the following: (a) conversion to another language, code or notation; and/or (b) reproduction in a different material form.

[00024] The foregoing description of the preferred embodiments of the invention has been presented for purposes of illustration and description. They are not intended to be exhaustive or to limit the invention to the precise form disclosed, and obviously many modifications and variations are possible in light of the above teachings. Such modifications and variations that are apparent to a person skilled in the art are intended to be included within the scope of this invention as defined by the accompanying claims.